INGENIEUR EN SCIENCES INFORMATIQUES

FINAL YEAR STUDY PROJET REPORT

Semantic head motion prediction in Virtual Reality videos

SUPERVISER: Frédéric Precioso, I3S Lab

CO-SUPERVISER: Miguel Romero and Lucile Sassatelli

STUDENT NAME: Mohamed YOUNES – SI5 Data Science Major

PROJECT TYPE: Scientific

Date: 9 October 2019 – 15 February 2020

PROJECT SUMMARY:

The massive adoption of virtual reality represents a major challenge: Internet streaming.
The problem is that what is displayed on the device (the field of vision - FoV) represents only a fraction of what is downloaded, and this wasted bandwidth is the price to pay for interactivity.[1]
To reduce the amount of data to be disseminated, recent work has proposed sending only the area of the video sphere corresponding to FoV in high quality [2-4].
These approaches require knowing in advance the position of the user's head, when sending the content from the server. It is therefore crucial for a 360 ° video streaming system to integrate a precise head movement predictor, capable of periodically informing, depending on the past trajectory and the content, that the user will be likely to watch at beyond the horizon.
To solve the problem of extracting valuable information from content, we propose to integrate latest generation information from visual capture in Deep Learning (DL) models.
The goal of the project is to re-implement one of the solutions to this problem, that was presented in the Conference on Computer Vision and Pattern Recognition 2018 (CVPR18), which claimed to outperform the existing methods and compare it to the solution proposed by the team of M. Romero & Al.

ACKNOWLEDGMENTS:

First, I would like to thank the following people for their help and valuable advice:

- Romero Rondón Miguel, PhD student and my co-supervisor of the project who has always been attentive and gave me good advice for data processing and the construction of neural network architecture and supervised me directly throughout the whole project.

- Sassatelli Lucile, Professor and member of SIS Team at I3S, on the data preprocessing and for helping me to get access to a server in the I3S Lab, so that I could generate data and train my model on a powerful machine.

Content :

I. Introduction:

As part of my engineering studies at Polytech Nice Sophia Antipolis, I carried out a graduation project entitled: Semantic head motion prediction in 360. This project took place under the supervision of Frédéric Precioso, Lucile Sassatelli and Miguel Romero, members of the SPARKS and SIS teams at the Laboratory of Computer Science, Signals and Systems of Sophia Antipolis (Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis) I3S between October and February 2019/20.

During this project, I had the opportunity to preprocess a dataset that contains 360° videos and to re-implement a solution introduced in CVPR18 which aims to predict user head movements wearing a VR headset, based on his movements in the past and the saliency elements of the scene in the video. This solution uses neural networks to solve this regression problem and will be compared to other methods in terms of performance and time costs.

II. Context and Subject presentation:

There is an emerging interest in viewing 360◦ VR contents in head-mounted displays (HMD) in place of a rectangular one on the screen. Commodity omnidirectional cameras such as Google Jump, Nokia OZO, Facebook 360, etc. are readily available to generate high quality 360◦ video contents and provide viewers with an immersive viewing experience. Social media platforms including YouTube and Facebook also support viewing 360◦ videos with or without HMDs. Same as 2D images and videos, the most intriguing problem in 360◦ videos, from both the commercial and technology perspectives, is to determine where a user would look at in the video. A successful solution will greatly benefit VR content creation. [5]

The link between the CVPR18 paper's task and 360◦ video setup: In gaze prediction on standard videos, users PASSIVELY watch the videos. In 360◦ immersive videos, users can ACTIVELY rotate the heads and body and decide where he looks. Previous saliency detection* in 360◦ videos watches STILL scene, so they can directly collect the eye fixation of all participants at the same scene for generating ground truth. The scene in this case is DYNAMIC, for each frame, where a participant looks depends on its starting point and his decision on movement direction. So, it is extremely difficult to annotate ground-truth for saliency detection. This task is doable because we can collect the ground-truth, and this task is useful for many applications. To decrease the amount of data to stream, a solution is to send in high resolution only the portion of the sphere the user has access to at each point in time, named the Field of View (FoV).[5]

The CVPR18 paper explores gaze prediction in 360◦ videos, aiming to study the user gaze behavior in 360◦ immersive videos. It shows the necessity and challenges for gaze prediction in this dynamic 360◦ immersive videos because of its importance for user behavior analysis while watching 360◦ VR videos and benefiting the data compression in VR data transmission [6]. Further, VR gaze prediction can also benefit applications beyond 360 videos: once we predict the viewing region of each participant viewer in upcoming frames, we can further improve user-computer interactions by tailoring the interactions for this specific viewer.

III. The solution's presentation :

In order to achieve this goal, many contributions suggest the use of saliency detection and gaze tracking. However small portions of these works focus on how a viewer interact with a 360° video. That's because it is difficult to track eyes that are behind an HMD, therefore it is not possible to track them using traditional camera-based tracking.

In the reference paper, they employ an emerging in-helmet camera system, a '7invensun a-Glass' eye tracker, that can capture eye locations for conducting gaze tracking when a user views a specific frame in 360◦ videos. Then they embed it into an HTC VIVE headset. With this device, they create a large-scale VR gaze tracking dataset by collecting the eye fixation of viewers with a gaze tracker deployed in an HMD when they watch 360◦ videos in a real immersive VR setting (the users also wear the earphones when they are watching VR videos).[5]

Next, with the dataset collected with an HTC VIVE headset and 7invensun a-Glass eye tracker, they conduct the gaze prediction in dynamic VR videos. They present a deep learning based computational model towards robust VR gaze prediction. The difference between watching a 2D videos and a 360° video is that a viewer will have a much higher degree of freedom when watching 360◦ videos. Specifically, they leverage an LSTM* module to estimate the viewer's behavior (watching pattern) under the fixed FoV. Nevertheless, a viewer is more likely to be attracted by salient objects characterized by appearance and motion, therefore they take into consideration the saliency, specifically, they consider the saliency at different spatial scales in terms of video contents in an area centered at current gaze point, the video contents in the current FOV, and the video contents in the whole 360◦ scene. Then they feed the images as well as their saliency maps at different scales into a CNN*. Then they combine the CNN features with LSTM features to predict the gaze displacement from current moment to next moment.[5]

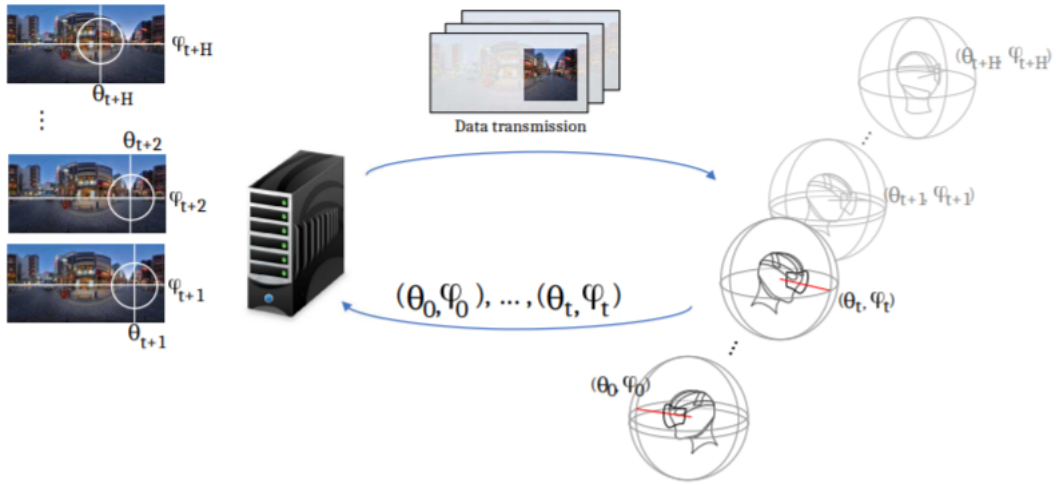


Figure 1: 360◦ video streaming principle. The user requests the next video segment at time t, if the future orientations of the user (ϑ_{t+1}, ϕ_{t+1}), ...,(ϑ_{t+H} , ϕ_{t+H}) were known, the bandwidth consumption could be reduced by sending in higher quality only the areas corresponding to the future FoV.

IV. Positioning of the solution against existing solutions:

Aside from the reference paper of CVPR18, there are several works that are relevant to the problem: PAMI18 [7], MM18 [9], ChinaCom18 [9] and NOSSDAV17 [10]. They all use both the history of past positions and knowledge of the video content. They differ on their objective of prediction; like predicting the future head position, gaze position or tiles in the FoV; The prediction horizon which is the range of the prediction in the future. Different types of input and input formats are considered (col. 5): some consider the positional information implicitly by only processing the content in the FoV (PAMI18), other consider the position separately, represented as a series of coordinates (CVPR18 in our case) or as a mask (e.g., MM18), with the last sample only (IC3D17) or various length of history, some extract features from the visual content by employing some pre-trained saliency extractors (e.g. NOSSDAV17, MM18) or training end-to-end representation layers made of convolutional* and max-pooling layers

(e.g., PAMI18). Finally, most of the methods but the first two in Figure 2 rely on deep learning approaches. There is also the way to which the information (video content and trajectory) are merged or fused. Because this type of information is sequential, all the works consider using deep recurrent neural networks, specially LSTMs, but the time of fusion depends on each work (col 6) .

Reference	Objective	Prediction horizon	Dataset	Inputs	LSTM before/after fusion
PAMI18 [9]	head coordinates	30ms	76 videos, 58 users	frame cropped to FoV	N/A (no fusion)
IC3D17 [20]	head coordinates	2s	16 videos, 61 users	Pre-trained sal. in FoV	N/A (no fusion, no LSTM)
ICME18 [21]	tiles in FoV	6s	18 videos, 48 users	Position history, users' distribution	N/A (no LSTM)
CVPR18 [10]	gaze coordinates	1s	208 videos, 30+ users	Video frame, position history as coordinates	before
MM18 [11]	tiles in FoV	2.5s	NOSSDAV17's dataset with custom pre-processing	Pre-trained sal., mask of positions	after
ChinaCom18 [12]	tiles in FoV	1s	NOSSDAV17's dataset	Pre-trained sal., FoV tile history	after
NOSSDAV17 [13]	tiles in FoV	1s	10 videos, 25 users	Pre-trained sal., FoV position or tile history	after

Figure 2: Existing dynamic head-prediction methods

Here is a description of each of these methods:

PAMI18: Xu et al. in [7] design a Deep Reinforcement Learning model to predict head motion. Their deep neural network only receives the viewer's FoV as a 42 × 42 input image and must decide to which direction and with which magnitude the viewer's head will move. Features obtained from convolutional layers processing each 360◦ frame cropped to the FoV are then fed into an LSTM to extract direction and magnitude. The training is done end-to-end. The prediction horizon is only one frame, i.e., 30ms. By only injecting the FoV, the authors make the choice not to consider the positional information explicitly as input. The PAMI18 architecture therefore does not feature any specific fusion module.

IC3D17: The strategy presented by Aladagli et al. in [11] simply extracts saliency from the current frame with an off-the-shelf method, identifies the most salient point, and predicts the next FoV to be centered on this most salient point. It then builds recursively.

ICME18: Ban et al. in [12] assume the knowledge of the users' statistics, and hence assume more information than our problem definition, which is to predict the user motion only based on the user's position history and the video content. A linear regressor is first learned to get a first prediction of the displacement, which it then adjusts by computing the centroid of the k nearest neighbors corresponding to other users' positions at the next time-step.

MM18: Nguyen et al. in [8] first construct a saliency model based on a deep convolutional network and named PanoSalNet. The so-extracted saliency map is then fed, along with the position encoded as a mask, into a double-stacked LSTM.

ChinaCom18: Li et al. in [9] present a similar approach as MM18, adding a correction module to compensate for the fact that tiles predicted to be in the FoV with highest probability may not correspond to the actual FoV shape (having even disconnected regions). This is a major drawback of the tile-based approaches as re-establishing FoV continuity may significantly impact final performance.

NOSSDAV17: Fan et al. in [10] propose two LSTM-based networks, predicting the likelihood that tiles pertain to future FoV. Visual features extracted from a pre-trained VGG-16 network are concatenated with positional information, then fed into LSTM cells for the past M time-steps, to predict the head orientations in the future H time-steps. Similarly to MM18, the building block of NOSSDAV17 first concatenates flattened saliency map and position and feeds it to a doubly-stacked LSTM whose output is post-processed to produce the position estimate.

The CVPR18 work, which is the center of this project, was chosen as a reference among other works because they formulate the gaze prediction problem in the same way as the head prediction problem. Therefore, the supervisor team considered its architecture and compared it with their baselines for the original problem of gaze prediction.[13]

V. Problem formulation: [5]

The authors formulated the gaze prediction in VR problem as follows: Given a sequence of 360° VR video frames $V_{1:t} = \{v_1, v_2, \ldots, v_t\}$ where v_t corresponds to the t^{th} frame, and the gaze points of the p^{th} user corresponding to this video ($L^p_{1:t} = \{l_1, l_2, \ldots, l_t\}$ where $l_t = (x_t, y_t)$, x_t and y_t is the latitude and longitude of the gaze intersection on a 3D Sphere where $x_t \in [0, 360]$, $y_t \in [-90, 90]$), then gaze prediction aims to regress the future gaze coordinates corresponding to the future T frames: L_i where i = t+1, ..., t+T.

The gaze pattern in future frames is related to multiple factors. On the one hand, gaze points are largely correlated with spatial saliency which can be inferred from image contents, and temporal saliency which can be inferred from the optical flow between neighboring frames. On the other hand, the users' history gaze path is also a key factor in predicting his/her future gaze point because different users have different habits in exploring a scene. For example, some users would look up and down frequently, and some users seldom look up or down. Another reason for leveraging users' history gaze path for gaze prediction is that users tend to explore the whole scene first by walking along the same longitude changing direction, i.e., walking from left to right or from right to left consistently rather than frequently changing the walking direction.

The gaze prediction is then formulated as a task of learning a nonlinear mapping function F which maps the history gaze path and image contents to the coordinates. The authors proposed to predict the displacement of gaze coordinates between the upcoming frame and current frame: $l_{t+1} - l_t$. Mathematically, the objective of gaze tracking is formulated as follows:

$$F^* = \arg\min_F \sum_{t=obs}^{obs+T-1} \|l_{t+1} - (l_t + F(\mathbf{V}_{t:t+1}, \mathbf{L}_{1:t}))\|^2$$

where obs is the number of observed frames. Two neighboring frames characterize the motion information (optical flow), and the next frame provides contents for saliency characterization. Then we use a deep neural network to model F (as shown in Fig. 3. Specifically, the network consists of a Trajectory Encoder module and a Saliency Encoder module, and a Displacement Prediction module. Next, we will detail these models sequentially.

1. Trajectory Encoder Module

Trajectory encoder module is designed to encode the history gaze path of a user. Viewers tend to explore the scene with the consistent direction along longitude. As for the gaze path direction along latitude, the gaze points usually are around the equator. In other words, the gaze path in history frames provides the clue for gaze prediction in future frames. Considering the good performance of LSTM networks for motion modeling [14], we also employ an LSTM network to encode the gaze pattern along the time. For each video clip, we sequentially feed the gaze points (l^p_t) corresponding to history frames in this video sequence into a stacked LSTM, and denote the output of stacked LSTM f^p_{t+1} at $(t+1)^{th}$ frames:

$$f^p_{t+1} = h(l^p_1, l^p_2, ..., l^p_t)$$

Here the function h(·) represents the input-output function of stacked LSTM. In addition, the function h is stacked LSTMs with 2 LSTMs layers, both with 128 neurons.

2. Saliency Encoder Module

Gaze points usually coincide with spatial salient regions and objects with salient motions (large optical flows). In other words, saliency provides an important cue for gaze prediction in future frames. Thus, it's proposed to incorporate the saliency prior regarding spatial saliency and temporal saliency which is characterized by optical flow features for gaze prediction. However, the saliency level of the same object at different spatial scales are different. So, the authors propose to calculate the saliency with a multi-scale scheme:

i) local saliency: the saliency of local patch centered at current gaze point.

ii) FOV saliency: the saliency of sub-image corresponding to current Field of View (FOV).

iii) Global saliency: the saliency of the global scene. In this implementation, we use the FlowNet2.0 [15] to extract the motion feature, and the input of FlowNet2.0 is the panorama images of two consecutive frames.

Then they propose to concatenate the RGB images, all spatial and temporal saliency maps, and feed them into an Inception-ResNet-V2 [16] to extract saliency features for gaze prediction. We denote z(·) represents the Inception V2 network, and denote S^p_{t+1} as all spatial and temporal saliency maps, and denote the saliency features as g^p_{t+1}, then g^p_{t+1} can be obtained as follows:

$$g^p_{t+1} = z(v_{t+1}, S_{t+1})$$

where z(·) represents the subnet from input to the layer after global pooling in Inception-ResNet-V2.

3. Displacement Prediction Module

The displacement prediction module takes the output of saliency encoder module and trajectory encoder module, use another two fully connected layer to estimate the displacement between the gaze point at time t + 1 and gaze point at time t:

$$\delta l^p_{t+1} = r([f^p_{t+1}; g^p_{t+1}])$$

where r(·) represents two connected layers. The function r contains two fully connected layers with 1000, 2 neurons, respectively. Once we get the location displacement, we can compute the gaze coordinate l^p_{t+1} at time t+ 1: $l^p_{t+1} = l^p_t + \delta l^p_{t+1}$. We train the model by minimizing this loss across all the persons and all video clips in the training-set.

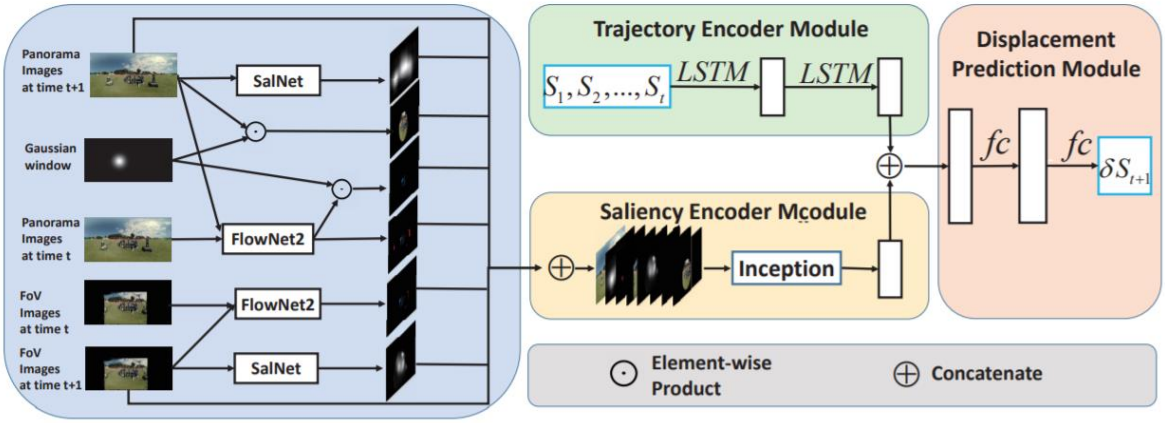


Figure 3: The architecture of the proposed deep neural network

VI. 360◦ Videos Dataset:

Some 360◦ video datasets have been created for viewer behavior analysis in VR and based on the ground truth annotated in these datasets, they can be categorized into head movements analysis-based datasets and gaze tracking based datasets. Head movements datasets only record the movements of heads, but even the heads are still, the viewers' eyeballs are still moving, that is, viewers still actively search the environment in their Field of View (FoV). So, datasets in this category cannot provide the detailed eye movement information, and the datasets in this category are usually used for data compression for VR videos. In contrast, eye tracking based VR datasets provide the gaze points (eye fixation) at a different time.

The dataset used and annotated by the authors consists of 208 high definition dynamic 360◦ videos collected from YouTube, each with at least 4k resolution (3840 pixels in width) and 25 frames per second. The duration of each video ranges from 20 to 60 seconds. The videos in this dataset exhibit a large diversity in terms of contents, which include indoor scene, outdoor activities, music shows, sports games, documentation, short movies, etc. Further, some videos are captured from a fixed camera view and some are shot with a moving camera that would probably introduce more variance in eye fixation across different users. They used an HTC

VIVE as their HMD to play the 360◦ video clips, a ’7invensun a-Glass’ eye tracker is mounted within the HMD to capture the gaze of the viewer. 45 participants (25 males and 20 females) aging from 20 to 24 is recruited to take part in the experiment. All participants were reported normal or corrected-to-normal vision in the HMD setting and were instructed to freely explore in the video scene.[5]

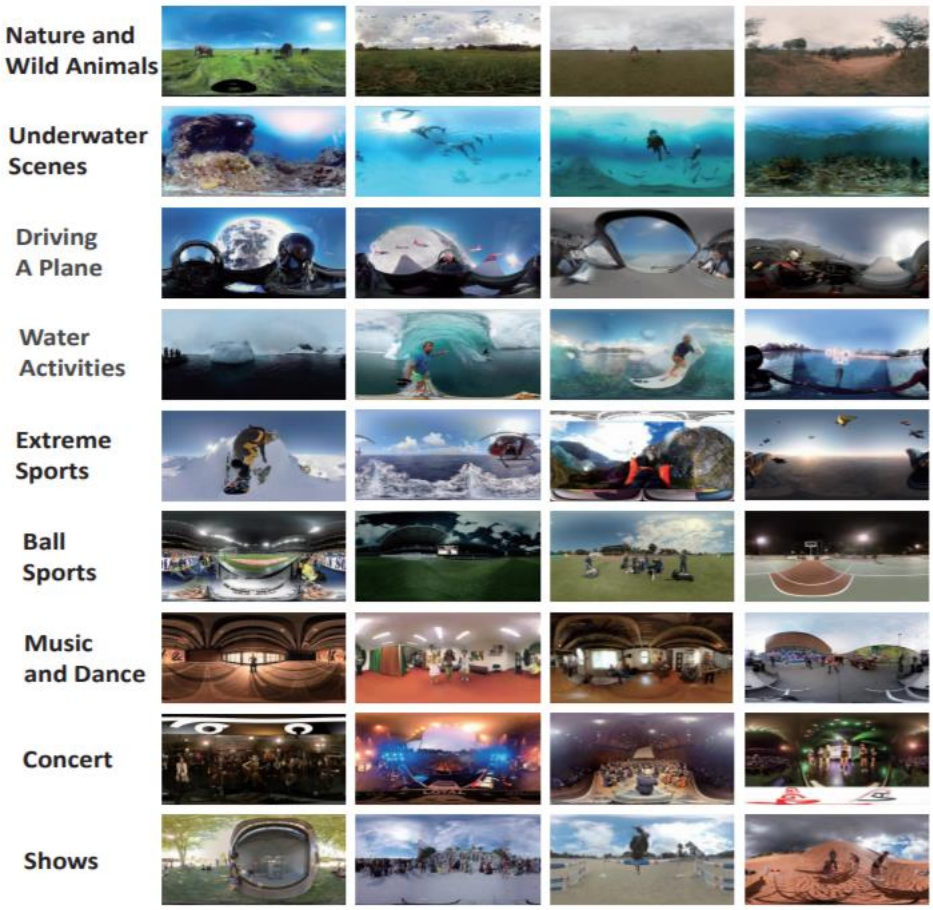


Figure 4: Examples of dataset videos

VII. The process of my work:

In this part, I will explain the different steps that I followed to get to the final product, that is to re-implement the work of the CVPR18 paper.

1. Extracting saliency elements from video frames:

The first thing is to extract the salient regions from the video frames since salient objects easily attract viewers' attention. In this case, saliency is related to both appearance and motion of the objects; Therefore, we could divide saliency elements into spatial and temporal saliency:

Figure 5: Example 360° video frame

- Optical flow of a frame with FlowNet2:

Viewers sometimes are attracted by moving objects, so temporal saliency is also an important factor for gaze tracking. Thus, we calculate the optical flow using FlowNet2.0, and the input of FlowNet2.0 is the panorama images of two consecutive frames.

I learned how to generate optical flow using flownet2-pytorch implementation. I worked on Google Colab since the flownet2 works only on GPU, and I modified their code so that it works for the Colab's GPU and loads the dataset videos (However, the dataset images files were in jpg format while the flownet2 works on PNG images, so I had to do some conversion).

This process did not generate optical flow images directly, only vectors that correspond to optical flow in each pixel with .FLO extension, therefore I used another package named Flowiz that transform these vectors into actual images:

Figure 6: Applying FlowNet2 on the previous frame

- Saliency detection with SalNet:

Saliency detection assumes those more salient regions attract viewers attention. So, we use the SalNet, which is a state-of-the-art saliency detection method, to calculate the spatial saliency map for each panorama image.

I worked on SalNet. I changed some lines in their code, so that it works for our dataset and because there were incompatibilities with the newest versions of some libraries:

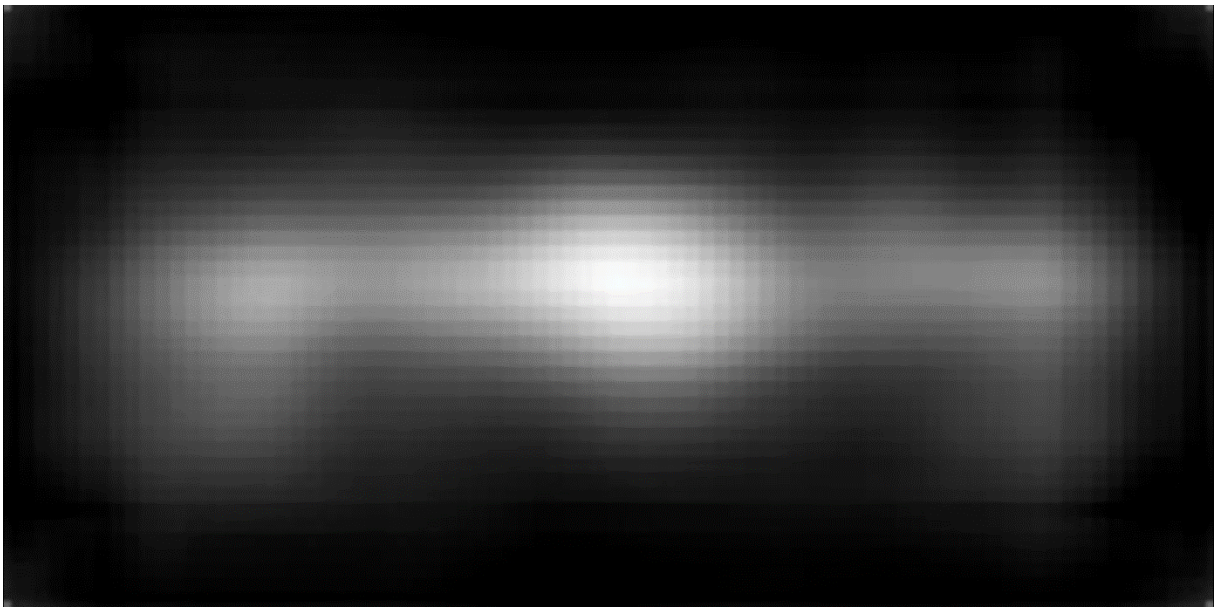


Figure 7: Applying SalNet on the previous frame

- FOV mask for local saliency:

A viewer's gaze point at next frame depends on the coordinates of current gaze points. So, we first generate a Gaussian window centered at current gaze point and use it to do the inner product with panorama image and optical flow map and use it as the spatial and temporal

local saliency maps. It is worth noting that the local image patch is usually very small, and it usually doesn't contain a complete object. According to the authors, the gaussian based saliency approximation demonstrates effectiveness and efficiency. Further, Gaussian based local saliency is also on par with that of classical saliency detection-based solution.

Nevertheless, I worked on the FoV mask that uses a gaussian window. In the CVPR paper, there is no mention for its parameters so I had to find them; since we are talking about FoV, I supposed that the mask must keep the interior of the window, therefore the expected value is zero for the gaussian distribution, and for the standard deviation it must correspond to the radius in the FoV; so since FoV in a VR headset is about 100° and the panoramic view is 360° I could get the radius in pixels from the width of the frame. At the end I found that 85 is a good value for the standard deviation. Then, for each frame and each viewer I took his gaze position and multiply by the frame's dimensions, so that I get the position in the frame by pixels:

Figure 8: Applying gaussian mask on the previous frame

- Patched frames:

For local saliency, I also generated a patch centered at current gaze point for each frame that correspond to the FoV of the viewer. This patch will also be transformed into a saliency map and optical flow. For this I applied the module of Nitish Mutha[that generate the spherical-to-rectangular projection on the 3 previous frames: the initial frame, the saliency maps and the optical flows.

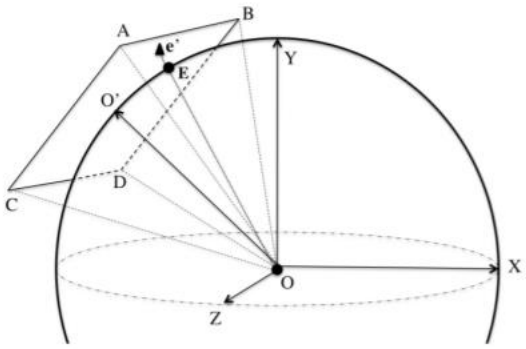


Figure 9: Spherical-To-rectangular projection

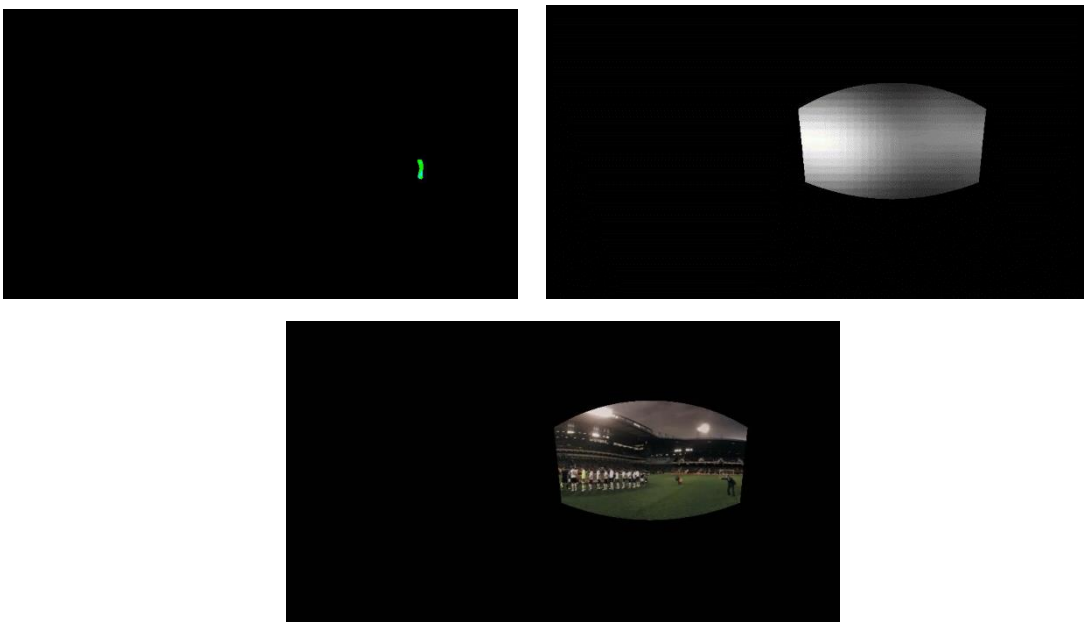


Figure 10: Applying projection on previous frame and its saliency map and optical flow

Figure 11: The pipeline of saliency detection in FoV

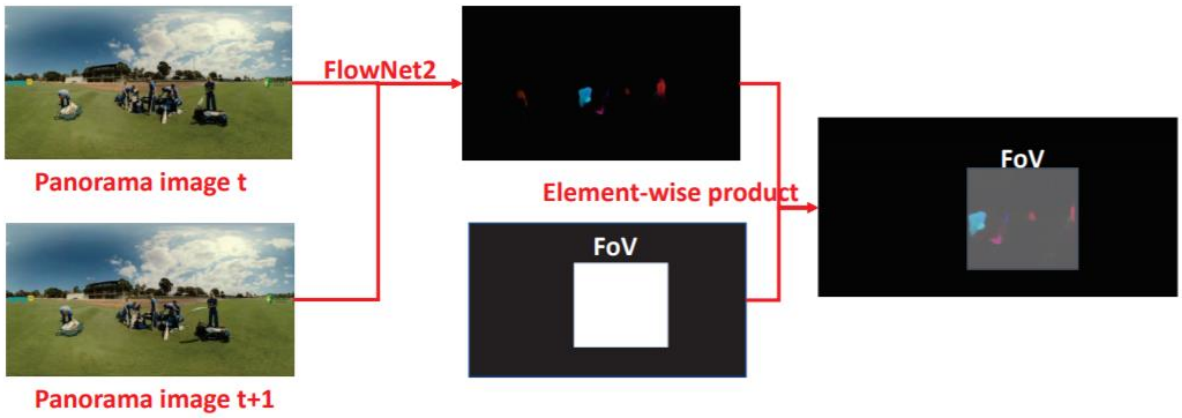


Figure 12: The pipeline of optical flow estimation in FoV

2. Neural Network Architecture:

I built the model of the deep neural network that corresponds to the proposed solution including all the modules (Trajectory Encoder Module, Saliency Encoder Module and Displacement Prediction Module) using the library of Keras.

I faced some problems in the implementation: I didn't have a good understanding of stacked-LSTM, but I got it right at the end (I stacked 2 LSTMs with 128 units each as mentioned in the paper).

For the saliency encoder, they concatenated 8 images to get the spatial and temporal saliencies, although they did not mention along which axis they concatenate, so I supposed that they will be concatenated by rows, therefore the input for this module would be (480*8, 960, 3). Then I added InceptionResNetV2 to this module as well. However, they mentioned that their output was the layer after global pooling, without saying which type of pooling they used, max pooling or average pooling. So, I chose the average pooling at the end since it's the popular one.

For the displacement encoder, I concatenated the outputs from previous modules and added two fully connected layers as mentioned:

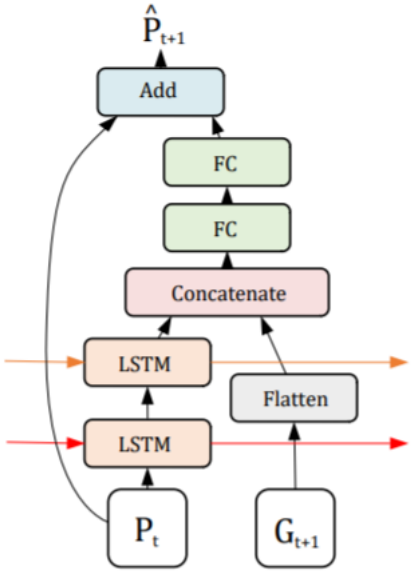


Figure 13: The building blocks in charge, at each time step, of processing positional information P_t and content information represented as saliency G_t, obtained from the saliency extractor module.

I want to note also that I used the functional API of Keras with Model class that let me have more control on the layers as well as having multiple inputs which is the case here.

For the global model I used Stochastic Gradient Descent as optimizer with different hyper-parameters as mentioned in the paper, because the one that was used (0.1) was causing the problem of exploding gradients, therefore I used a smaller value (0.01). For the loss function, I used Mean Squared Error.

There was something weird in the paper's model architecture: In the first part where they explain the different modules, they simply generate the displacement of the gaze position for the next frame and they add it to the previous gaze position to get the next frame's gaze position. However in their experimental setup part, they proposed to 'use the history gaze path in the first five frames to predict the gaze points in next five frames', which needs another type of architecture that could be serially replicating their initial architecture 4 more times or using sequence to sequence unit in the trajectory module with or without teacher forcing. Because of the ambiguity, I built two models: one that will predict only the next frame's gaze position using the architecture of the paper, and one that will predict the next 5 frames' gaze positions using sequence-to-sequence trajectory module without teacher forcing and using Keras functions like Lambda and Add to manipulate the output and move through each frame. Then I will compare both in terms of performance and time cost.

2. Resources issues:

I faced some problems when I was working with Google Colab that offers 12 Gb of GPU for 12 continuous hours. Because the dataset was huge and needed long time for extractions and transformation, I asked my supervisors to give me access to a server in I3S that I can use for my different tasks. After getting access to the server, I had to set it up with Docker, transfer all the files and images I worked with in Google Colab. In the beginning I faced some issues

and conflicts between packages since the version of ubuntu in the server is not the same as in Colab, also the installation of Caffe for the saliency detection was more complicated which took me some time to solve it. At the end I managed, and everything worked fine.

3. Dataset generation:

For the data preprocessing, in order to build the training set. I started with converting initial dataset videos into frames that will be transformed then fed to the network as input. For that I used a Linux program called ffmpeg . Like in the paper, I downsampled one frame from every five frames so that the interval between two neighboring frames in our experiments corresponds to 0.2 seconds, and such setting makes the gaze prediction task more challenging than that for the neighboring frames in original videos.

Then I worked on the gaze files: I extracted the gaze history for each (user, video) that are couples (x, y) where x is the latitude and y is the longitude, normalized with respect to the resolution of the images. They have the left bottom corner as position of the origin (0, 0), Then I grouped them in windows of 5 frames so that for each video and at an instant t, I have the 5 previous gazes positions that will be used in the trajectory encoder and the next 5 gaze positions that will be used as ground truth for the prediction.

The next step was to generate patched frames for optical flow, which needed the original optical flow frames that I had already generated in Colab, then apply the FOV projection to get the patched frames.

At last, I had all the pieces to construct the dataset that will be fed to the network :

- The panoramic frames at time t+1
- The FOV frames with gaussian mask
- The saliency maps for the frames
- The optical flow of frames at time t+1
- The FOV optical flow with gaussian mask at time t+1
- The patched frames at time t+1
- The optical flow of patched frames at t+1
- The saliency map of the patched frames at t+1
- The gaze positions for previous and next 5 frames at each instant t

This is the step where I was struggling much, because we were dealing with 8 images for each frame, user, video. The complexity was just insane! In addition, because of the long processing of previous steps(optical flow and patched frames), that used only 2 of the available 20 cores of the server, I was thinking about parallelizing the processes using some python libraries like multiprocessing and Joblib.

After generating the different spatial and temporal saliency elements. The goal was to concatenate them into files along with trajectory gazes. For that, I created npz compressed array files (in order to save space in disk). Each npz file contains 3 arrays: The concatenated images in an array name ‘saliency’, The previous 5 frames’ gaze positions in an array named ‘trajectory_x’, The next 5 frames’ gaze positions in an array named ‘trajectory_y’. the images

were normalized in the same process. The npz file were stored in the format a_b_c.npz where a is the video number or id, b is the frame number and c is the user number.

The big issue was that it needed a long time to generate, because of the huge number of triples (video, user, frame), about 1 152 761; each piece of data contains the saliency information (8 images of spatial and temporal saliency) as well as the trajectory information (last 5 frames' gaze positions + next 5 frames' gaze positions).

At first, I tried to store the data in npy file format, however the size of each file was about 80 MB generated in 0.04 second, which would need more than 92 TB in total that is not available in the server. This is why I used the compressed format npz that gives each file 3MB but sacrificing the time of data generation because of compression (2.3 seconds). The whole process took about 2 weeks to finish, knowing that I used all the 20 cores available to parallelize the process and also, I used the Cupy library instead of Numpy since it utilizes the GPUs for array computations giving a 5x boost.

- Data generator:

I used a custom data generator to feed data to the network as batches of a specific size. It takes the list of npz files and for each frame it generates arrays yields the appropriate arrays for the network input and output. In the case where I use the model that predict the next 5 gazes' positions, it searches for the next 4 frames files in the dataset directory to concatenate with the actual frame.

4. Training phase:

The Neural Network has 2 inputs that takes 'saliency' and 'trajectory_x' arrays. 'trajectory_x' will go through 2 lstm encoder-decoder while 'saliency' will go through inception resnet v2. The network will iterate through 5 consecutive frames to predict the next 5 gaze positions and compare them with 'trajectory_y', so there are lambda functions in the model that take a specific frame at time t and other functions that add the resulting predicted variation to the next gaze position and so on.

5. Issues in the Training:

I faced many problems in the part of feeding the data to the neural network as well. First, I made a mistake when building the network, that is creating many dense layers for each frame instead of reusing and sharing them. This mistake was the reason that caused Tensorflow library to alert about issues with the topological order of the network.

Second, there was another issue with the excessive memory usage; I found out when monitoring the GPUs with nvidia-smi that the network wasn't loaded in the GPU at all, and that was clear from Tensorflow alerts when it shows skipping registering gpu devices. Therefore, I deduce the model was loaded in ram and computations were managed by CPU instead. The reason behind that was an incompatibility between versions of CUDA, CuDNN, Tensorflow and Keras, which leads to not recognizing the GPUs. I fixed that problem and the model was finally mapped in the GPU while RAM is used only to read batches of data.

Even when loading the model in the GPUs, it was so big to be contained, so I decided to freeze layers of the inception network and use only the initial weights of 'imagenet' for the saliency module.

Third, at some point of the training, the loss cost becomes nan, meaning that there are data that were corrupted or non-numerical values. To solve this problem, I checked for the corrupted files and those containing non-numerical values, which took about 3 days since the dataset is huge and because of high dimensions ((8, 3840, 960, 3) for each file): I found about 300 files with these problems and deleted them. However, the problem of NaN loss wasn't solved after this, therefore I was convinced that it's due to "The exploding gradients". With some research I found out two solutions: Either use weight clipping values to a range or reduce the learning rate of the optimizer; Because I didn't know the range to limit the gradients I didn't use weight clipping, instead I decreased the learning rate from 0.1 to 0.01;

As result, the loss value became stable: The 0.1 learning rate was very high and many people reported that they faced the same problem because of it, so I reduced it even though the paper mentioned that they used that value.

6. Experiments and results:

Because I didn't have time to train the model on a window of 5 frames, I decided to train the model with only one frame window (a prediction horizon of 0,20 second).

After training the model, I loaded it with the learned weights and I applied it on a sampled video for a certain user from the train set and I computed the mse loss value (0,01) obtained after prediction, and did the same for a sampled video from the test set (which means that it's a new video for the trained model), the loss value was 3 times higher 0.03.

For the loss function for training and validation; you can notice that the train loss was converging very slowly, it could be because I set a small learning rate 0.01 but don't forget that using the same learning rate as in the paper 0.1 caused exploding gradients, it could only due to the complexity of the input itself. The validation loss was also converging but unstably. The best valid loss I obtained was of 0.007 and the learning stopped at train loss of 0.002 at epoch 56:

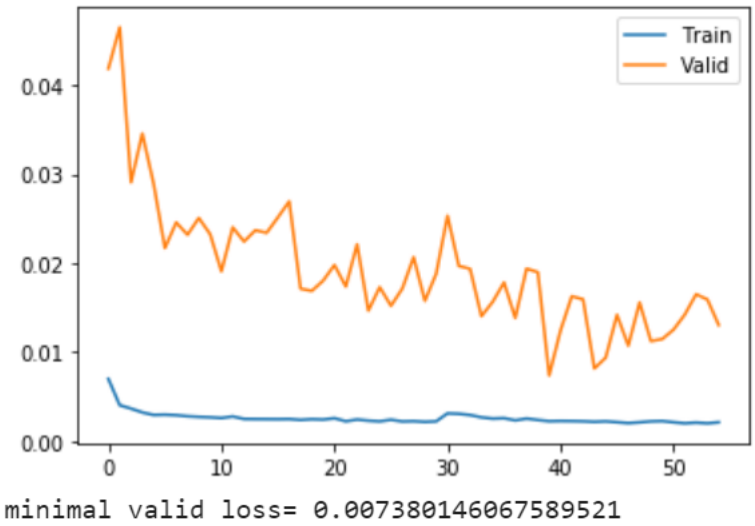


Figure 14: Train and valid loss for the one frame prediction model

Next, in order to visualize the result on the videos, I tried to represent the ground truth position of the gaze by a red dot and the predicted gaze by a blue one, then generate a video using all the frames. You can see that the results are not very good (you can't expect a lot from a model that skipped 4 next frames in the training) but the model is reacting to moving and salient objects and there are times when the two dots met:

Figure 15: Gaze prediction (red dot is ground truth and blue dot is predicted)

Figure 16: prediction on the previous frame (belonging to training set)

Figure 17: gaze prediction on a frame from testing set

However when it comes to execution time, the average running time was about 0,4s which makes the model non practical at all since it has a horizon of 0,2s; That put doubts on the model with 5 frame window (horizon of 1s) if it will do better in terms of performance and execution time knowing that the paper's results were 47ms with GPU and 466 ms with CPU.

VIII. Conclusion:

- There were a lot of ambiguities in the paper when I tried to replicate it, and I found a lot of inconsistencies:

- The authors didn't explain how to get the longitude and latitude from the estimate displacement, in the paper they show it as a simple addition, without considering the cyclical features of longitude and latitude.
- The remapping of the FoV into the equirectangular is not that easy and they didn't provide any code for that, to reproduce the result I had to use the code of 'Nitish Mutha' which is not guaranteed to give the same result.
- They did not specify how the prediction window moves, if in a sliding window or not.
- They didn't explain how to perform the concatenation of the input images, horizontally, vertically or one per channel.
- They didn't define the parameters of the gaussian window, especially the standard deviation.
- They didn't show if there is any activation function used to limit the output values, they mentioned only using two fully connected layers at the end of the network without specifying if the activation was Relu of Tanh functions.
- The learning rate used in the paper 0.1 led us to the exploding gradient problem.
- The batch size and the number of epochs lead to a very long training time that wasn't mentioned in their work.
- The execution-time delay due to the complexity of the model.
- The uncertainty about if they used a Seq2Seq architecture or not.
- The performance improvement when using the visual input may come from the gaussian maps and not from the saliency information. The gaussian maps give information about the future head positions (information that we wouldn't have in a real environment).

- What remains undone for this project is:

- Training the full model predicting the next 5 frames' gazes' position, even though I doubt that it will do better in terms of time cost, since it needs to deal with 4 more frames and their saliency elements.
- Evaluating the results using another metric called Mean Intersection Angle Error that compares predicted and ground truth gazes in terms of angles instead of distance on panoramic view. That way we can be sure of the inconsistencies with the paper since this was the used metric.
- Reviewing the different choices done through the experimentation and that led to contradicted results with what was claimed in the paper. For example, should we resize the initial frames in order to reduce complex and huge dimensions ? That could reduce the time cost and fasten calculations.

IX. References:

[1] Corbillon, X., Simon, G., Devlic, A., & Chakareski, J. (2017, May). Viewport-adaptive navigable 360-degree video delivery. In 2017 IEEE international conference on communications (ICC) (pp. 1-7).IEEE.

[2] Qian, F., Ji, L., Han, B., & Gopalakrishnan, V. (2016, October). Optimizing 360 video delivery over cellular networks. In Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (pp. 1-6). ACM.

[3] Xiao, M., Zhou, C., Swaminathan, V., Liu, Y., & Chen, S. (2018, April). Bas-360: Exploring spatial and temporal adaptability in 360-degree videos over http/2. In IEEE INFOCOM 2018-IEEE Conference on Computer Communications (pp. 953-961). IEEE.

[4] Hristova, H., Corbillon, X., Simon, G., Swaminathan, V., & Devlic, A. (2018, August). Heterogeneous Spatial Quality for Omnidirectional Video. In 2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP) (pp. 1-6). IEEE.

[5] Y. Xu, Y. Dong, J. Wu, Z. Sun, Z. Shi, J. Yu, and S. GaoGaze. In Prediction in Dynamic 360◦ Immersive Videos

[6] M. Yu, H. Lakshman, and B. Girod. A framework to evaluate omnidirectional video coding schemes. In Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on, pages 31–36. IEEE, 2015.

[7] M. Xu, Y. Song, J. Wang, M. Qiao, L. Huo, and Z. Wang, “Predicting head movement in panoramic video: A deep reinforcement learning approach,” IEEE Trans. on PAMI, 2018.

[8] A. Nguyen, Z. Yan, and K. Nahrstedt, “Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction,” in ACM Int. Conf. on Multimedia, 2018, pp. 1190–1198.

[9] Y. Li, Y. Xu, S. Xie, L. Ma, and J. Sun, “Two-layer fov prediction model for viewport dependent streaming of 360-degree videos,” in EAI Int. Conf. on Communications and Networking (ChinaCom), Chengdu, China, Oct. 2018.

[10] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, “Fixation prediction for 360 video streaming in head-mounted virtual reality,” in ACM NOSSDAV,2017, pp. 67–72

[11] A. D. Aladagli, E. Ekmekcioglu, D. Jarnikov, and A. Kondoz, “Predicting head trajectories in 360◦ virtual reality videos,” in IEEE Int. Conf. on 3D Immersion (IC3D), 2017.

[12] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Wang, “Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming,” in IEEE Int. Conf. on Multimedia and Expo (ICME), 2018.

[13] M. Romero, L. Sassatelli, R. Pardo, F. Precioso, “Revisiting Deep Architectures for Head Motion Prediction in 360◦ Videos” in Computer Vision and Pattern Recognition, 2020.

[14] Y. D. B. Z. Hang Su, Jun Zhu. Forecast the plausible paths in crowd scenes. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, pages 2772–2778, 2017.

[15] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. 2016

[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In AAAI, volume 4, page 12, 2017.

[17] NitishMutha’s code for spherical-to- equirectangular projection: https://github.com/NitishMutha/equirectangular-toolbox

Tutorials and articles that I read during the project:

Basics of Keras: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

How to use the Functional API of Keras: https://machinelearningmastery.com/keras-functional-api-deep-learning/

LSTM with Keras: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Deeper into LSTMs - Sequence to Sequence models with Keras: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

tutorial on FlowNet2: https://towardsdatascience.com/generating-optical-flow-using-nvidia-flownet2-pytorch-implementation-d7b0ae6f8320

the code of FlowNet 2.0: https://github.com/NVIDIA/flownet2-pytorch

The code for SalNet : https://github.com/imatge-upc/saliency-2016-cvpr
Creating the gaussian window: https://www.w3resource.com/python-exercises/numpy/python-numpy-exercise-79.php

Inception ResNet V2 on Keras: https://keras.io/applications/#inceptionresnetv2

FC and LSTM layers: Here a quick introduction to LSTMs in Keras: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47

Using Docker in the server for preprocessing and training: https://docs.docker.com/get-started/

Using Cupy instead of numpy to speed Data generation up: https://towardsdatascience.com/heres-how-to-use-cupy-to-make-numpy-700x-faster-4b920dda1f56

Create a custom Data Generator: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

My github repo containing all the script for generating the dataset and the architecture of the network as well as the results on some videos: https://github.com/Polytech-PFE2019/pfe-2019-pfe2019-015

** The illustrations in the figures 1, 2, 3, 4, 9, 11, 12 and 13 are from either the reference paper or the supervisor team's paper.

X. Appendices

1. Saliency:

It is the process of applying image processing and computer vision algorithms to automatically locate the most "salient" regions of an image.

In essence, saliency is what "stands out" in a photo or scene, enabling your eye-brain connection to quickly (and essentially unconsciously) focus on the most important regions.

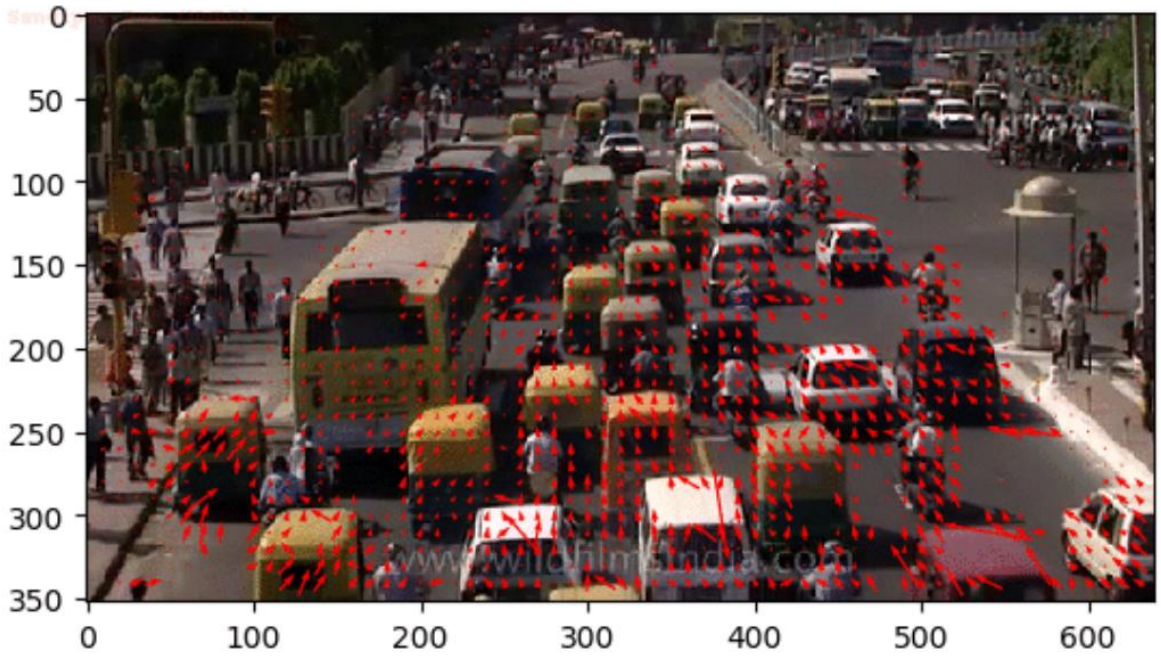
For example — consider the figure at the bottom where you see a soccer field with players on it. When looking at the photo, your eyes *automatically* focus on the players themselves as they are the most important areas of the photo. This automatic process of locating the important parts of an image or scene is called *saliency detection*:

Saliency detection is applied to many aspects of computer vision and image processing, but some of the more popular applications of saliency include:

- **Object detection** — Instead of exhaustively applying a sliding window and image pyramid, only apply our (computationally expensive) detection algorithm to the most salient, interesting regions of an image most likely to contain an object
- **Advertising and marketing** — Design logos and ads that "pop" and "stand out" to us from a quick glance
- **Robotics** — Design robots with visual systems that are similar to our own

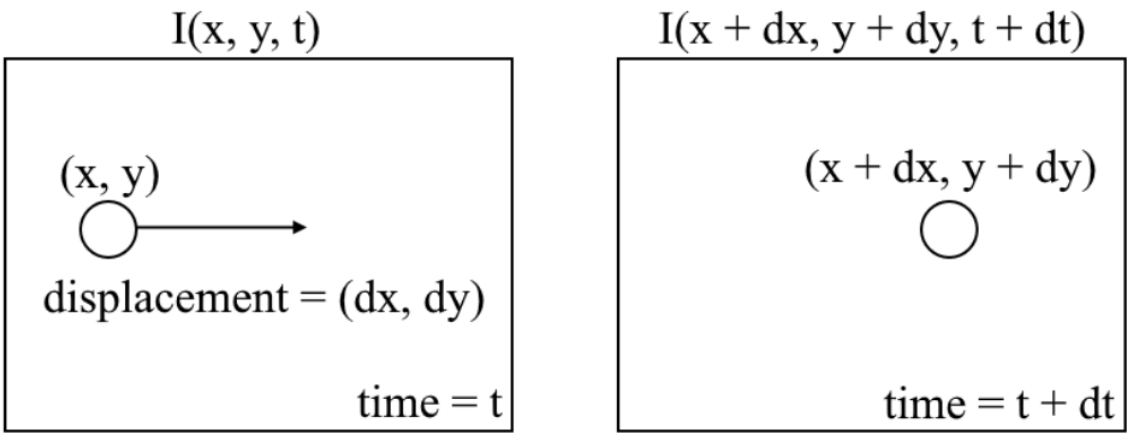
2. Optical Flow:

For processing real-time video input, most implementations of techniques like object detection for detecting instances of objects belonging to a certain class and semantic segmentation for pixel-wise classification, only address relationships of objects within the same frame (x,y) disregarding time information (t). In other words, they re-evaluate each frame independently, as if they are completely unrelated images, for each run. However, what if we do need the relationships between consecutive frames, for example, we want to **track the motion of vehicles across frames** to estimate its current velocity and predict its position in the next frame?



Sparse optical flow of traffic (Each arrow points in the direction of predicted flow of the corresponding pixel).

Optical flow is the motion of objects between consecutive frames of sequence, caused by the relative movement between the object and camera. The problem of optical flow may be expressed as:



Optical flow problem

where between consecutive frames, we can express the image intensity (I) as a function of space (x,y) and time (t). In other words, if we take the first image I(x,y,t) and move its pixels by dx,dy) over t time, we obtain the new image I(x+dx,y+dy,t+dt).

First, we assume that pixel intensities of an object are constant between consecutive frames.

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

Constant intensity assumption for optical flow

Second, we take the Taylor Series Approximation of the RHS and remove common terms.

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \dots$$

$$\Rightarrow \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

Taylor Series Approximation of pixel intensity

Third, we divide by dt to derive the optical flow equation:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

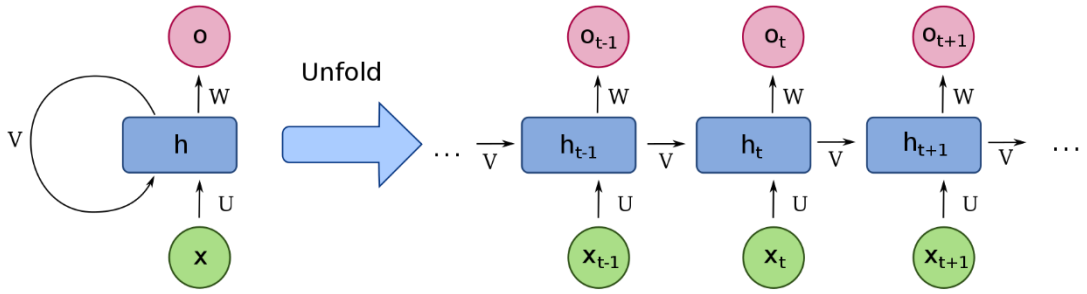
Optical flow equation

where u=dx/dtu=dx/dt and v=dy/dtv=dy/dt.

dI/dx, dI/dy, and dI/dt are the image gradients along the horizontal axis, the vertical axis, and time. Hence, we conclude with the problem of optical flow, that is, solving u(dx/dt) and v(dy/dt) to determine movement over time. You may notice that we cannot directly solve the optical flow equation for u and v since there is only one equation for two unknown variables.

3. Recurrent neural networks:

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

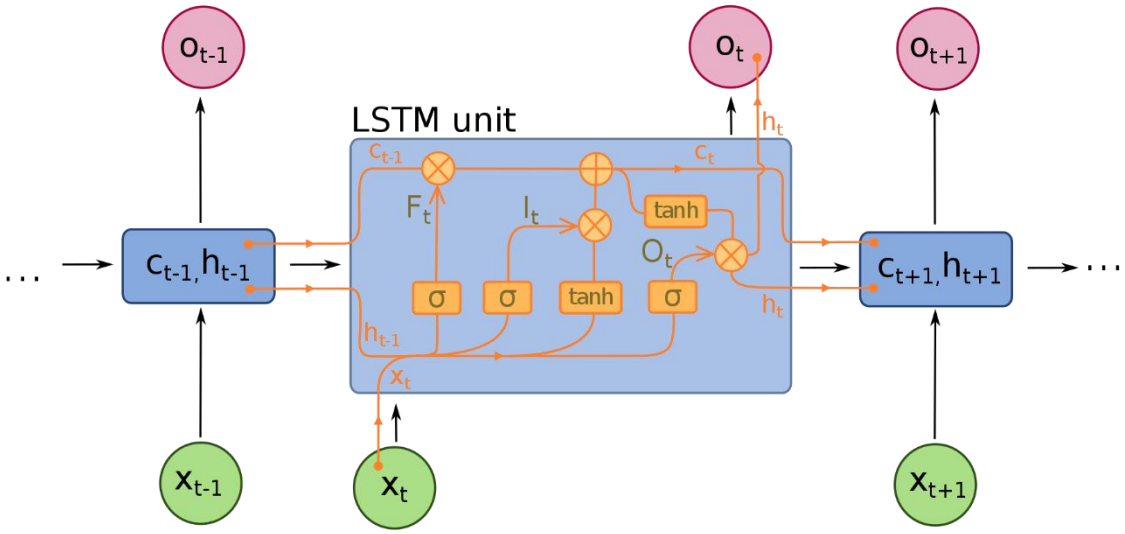


4. LSTM:

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget" gates. LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. Problem-specific LSTM-like topologies can be evolved. LSTM works even given long delays between significant events and can handle signals that mix low and high frequency components.

Many applications use stacks of LSTM RNNs and train them by Connectionist Temporal Classification (CTC) to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences. CTC achieves both alignment and recognition.

LSTM can learn to recognize context-sensitive languages unlike previous models based on hidden Markov models (HMM) and similar concepts.



5. Exploding gradients:

An error gradient is the direction and magnitude calculated during the training of a neural network that is used to update the network weights in the right direction and by the right amount.

In deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients. These in turn result in large updates to the network weights, and in turn, an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1.0.